Search Engine Style Query Handling

June 16, 2010

The pyparsing library is a terrific way of parsing and executing grammars.

It's yet another reason I continue to work in more and more in Python at the expense of Common Lisp, despite Python's pedigree as a language for teaching programming to the uninitiated.

Among the examples in the wiki is searchparser.py which adapts pyparsing to the task of handling full-text queries in the way most search engines do: exact phrases in quotes, multiple phrases grouped by parentheses, compound queries joined by "AND", "OR", and "NOT" operators recursively, etc.

After experimenting with it for a while, there was one change I made which seemed an improvement over the original:

The evaluateQuotes() method takes an argument, which represents the string containing an exact phrase defined by quotes in the original query.

```
def evaluateQuotes(self, argument):
    """Evaluate quoted strings

    First is does an 'and' on the indidual search terms, then it asks the
    function GetQuoted to only return the subset of ID's that contain the
    literal string.
    """
    r = Set()
    search_terms = []
    for item in argument:
        search_terms.append(item[0])
        if len(r) == 0:
            r = self.evaluate(item)
        else:
            r = r.intersection(self.evaluate(item))
    return self.GetQuotes(' '.join(search_terms), r)
```

As the documentation says, it looks up each individual word of the phrase first, and then invokes GetQuotes() with two parameters: the entire phrase string, and the result of all the individual lookups which were common to every word in the phrase.

If, however, the underlying data structure supports the idea of finding an exact phrase within a block of text efficiently, then there is no need to lookup each word of the larger phrase individually.

So evaluateQuotes() can be simplified to:

```
def evaluateQuotes(self, argument):
    """Evaluate quoted strings by invoking GetQuotes() on the entire quoted term"""
    search_terms = []
    for item in argument:
        search_terms.append(item[0])
    return self.GetQuotes(' '.join(search_terms))
```

The signature for the GetQuotes() method becomes:

```
def GetQuotes(self, search_string):
```

And finally, implementing GetQuotes() is simple, i.e., all it has to do is return a set containing occurences of the exact search_string within the database.
